第3章

部分リコンフィギャラブル機能の評価

──回路構成や回路定数の変更をマイコンから行う

野村英方

プログラマブル・デバイスの特徴を生かした使いかたの一つに，システム動作中のリコンフィグレーション（回路の再構成）がある．アナログ回路では，処理しようとする信号に合わせて回路構成や回路定数を最適に切り替えたい場合も考えられる．ここでは，FPGAに実装したマイコンを使ってアナログ・プログラマブル・デバイスの回路構成を部分的に変更する方法について解説する．（編集部）

アナログ・プログラマブル・デバイスは，アンプのゲインやフィルタ特性のみならず，回路構成の変更まで，LSIの内部だけで行うことができます．さらに一部のアナログ・プログラマブル・デバイスには，LSI動作時にLSIの一部分だけを書き換える機能があります．例えば複数の信号をマルチプレクサで切り替えて扱うようなアプリケーションにおいて，信号の切り替えに合わせて信号処理回路もいっしょに切り替える（まるごと書き換える）ような使いかたができます．

部分リコンフィギャラブル機能を持つデバイスの一つに，英国Anadigm社のFPAAがあります．FPAAでは，1クロックで回路を切り替えられます．回路の切り替えにあたっては，ROMからのコンフィグレーションではなく，外部マイコンなどを使って制御します．

この機能により，一つのLSIに複数の機能を持たせることができ，基板面積の縮小やコストダウンに役立ちます．

FPAAの部分リコンフィグレーション

FPAAの場合，回路の再構成には2通りの方法があります．いずれも，開発ツールの「Anadigm Designer2」から書き換え用のソース・コードを生成します．

● state-driven method

回路データをあらかじめ準備しておき，適当なタイミングで回路の置き換えを行います．例えばフィルタを発振器に置き換えるなど，回路そのものを変更できます．書き換える回路があらかじめ決まっているのであれば，この方法が便利です．

● algorithmic method

アナログ・ブロック（CAM）パラメータの変更や，マルチプレクス入力セルの切り替えを行う方法です．マイコンなどからパラメータ値を直接書き換えることができます．

例えば，現在ゲイン（利得）が5のアンプを，ゲインが6のアンプに変更することを考えます．ゲイン5とゲイン6の二つの回路をあらかじめ用意しておく方法ももちろん考えられます．しかし，ゲインがあらかじめ決められない場合もあるでしょう．algorithmic methodはこのようなときに，「ゲインを6にせよ」と命令することで回路の特性だけを変更できる機能です．パラメータ変更だけであれば，書き換え用のデータは小さくて済むため，制御時間も短くなります．

ただし，同じ機能でも設定するパラメータによって回路が切り替わる場合は，回路構成が変わることになるので，state-driven methodを使う必要があります．

FPAAを制御するためのCソース・コードは，「Anadigm Designer2」が生成してくれます．

部分リコンフィグレーション回路の設計

ここでは，FPAA（AN220E04）の部分リコンフィギャラブル機能をテストする回路を設計します．ソフト・マクロ

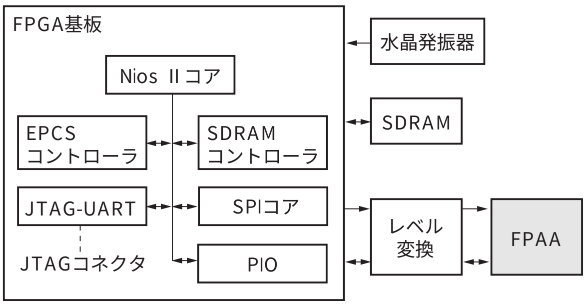
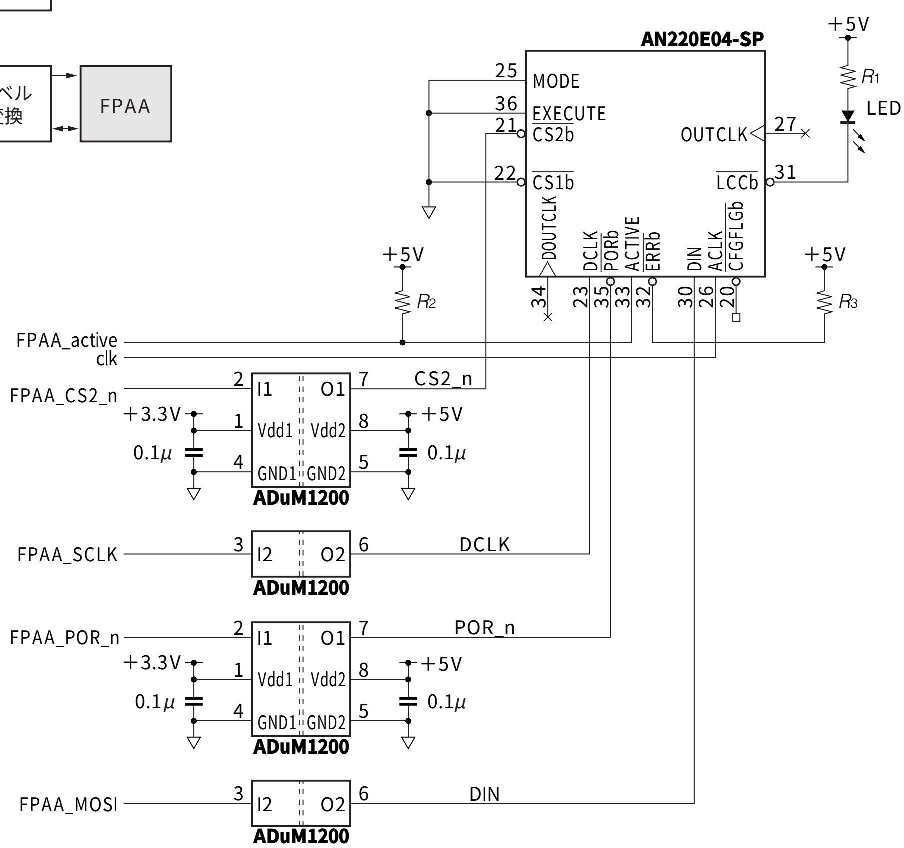


図1　部分リコンフィグレーション・テスト・システムのブロック図

FPAA（AN220E04）の部分リコンフィギャラブル機能をテストするシステムである．ソフト・マクロのCPUコアを実装したFPGAを使って制御する．FPGAは，Altera社のCyclone．本誌2003年10月号に付属していたFPGA基板を活用した．



▶図2
FPAA-FPGAインターフェース部の回路

FPAAとFPGAの間のインターフェースはSPIである．ほかに制御用のディジタル信号がある．FPAAは5V系のI/Oなので，レベル変換を行っている．

のCPUコアを実装したFPGAを使って制御します．FPGAとしては，米国Altera社のCycloneを使います．本誌2003年10月号に付属していたFPGA基板を活用しました．実装したCPUコアはNios IIです．

FPAAの回路は，マルチプレクス入力セルを切り替えるだけの簡単なものとしました．動作開始時にマルチプレクス入力セルのラインBを選択する回路とし，ラインA，C，Dと切り替えて出力します．

テスト・システムのブロック図を**図1**に示します．FPAAとFPGAの間のインターフェースはSPIとします．インターフェース部の回路を**図2**に，製作したテスト・システムを**写真1**に示します．

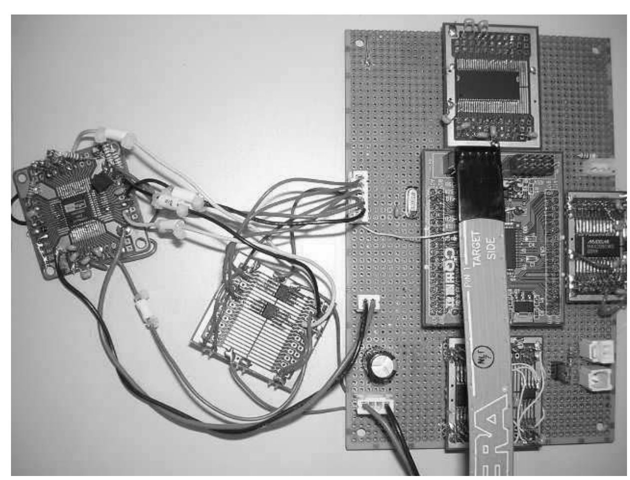
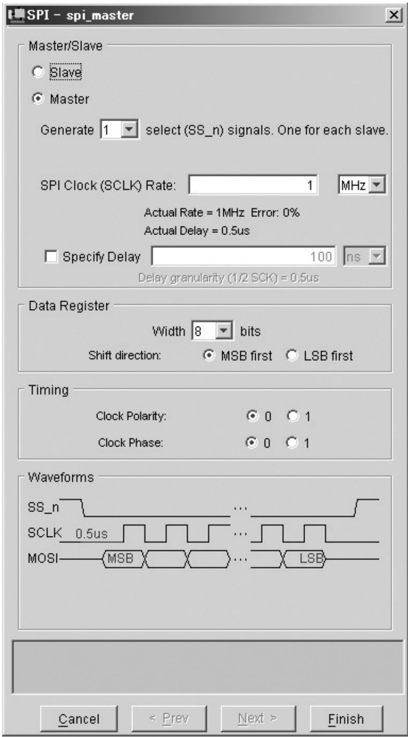


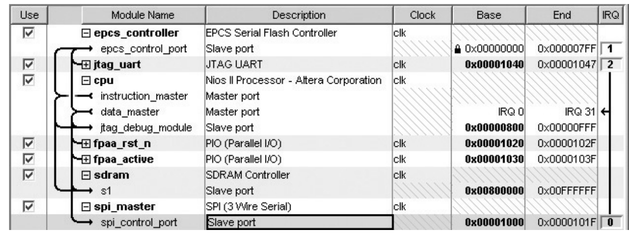
写真1　製作したテスト・システム

● 書き換え制御回路（FPGA部）の設計

書き換え制御を行うFPGAには，CPUコアを実装します．近年のFPGAはメモリ・ブロックを搭載していますが，小規模な品種ではCPU用の命令メモリとして使うには不十分です．そこで今回は，外部メモリとしてSDRAMを接続しました．



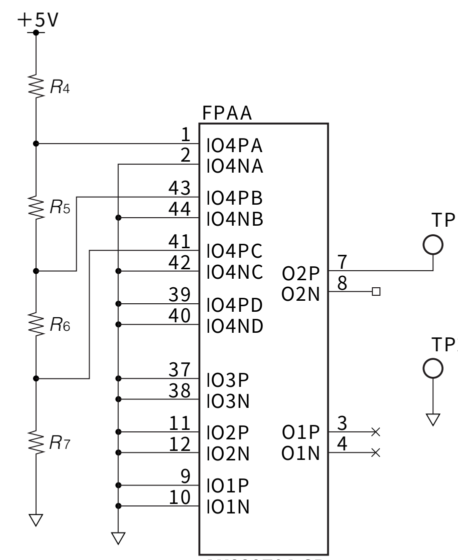
◀ **図3**
SPIコアの設定
Masterに設定する．SCLKは1MHz，データ・レジスタは8ビット．



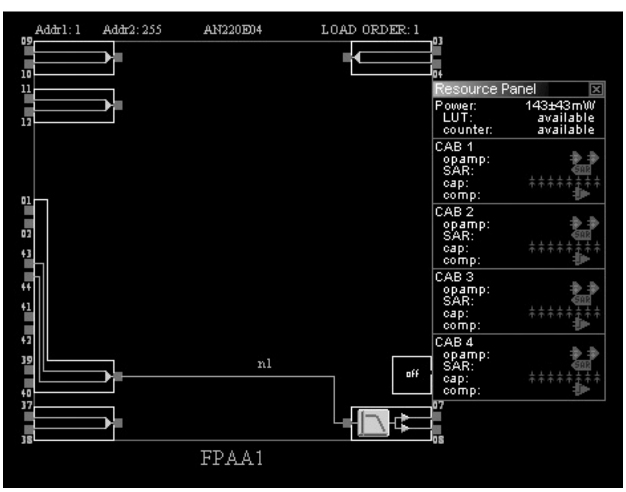
Use	Module Name	Description	Clock	Base	End	IRQ
☑	epcs_controller	EPCS Serial Flash Controller	clk			
	epcs_control_port	Slave port		0x00000000	0x000007FF	1
☑	jtag_uart	JTAG UART	clk	0x00001040	0x00001047	2
☑	cpu	Nios II Processor - Altera Corporation	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00000800	0x00000FFF	
☑	fpaa_rst_n	PIO (Parallel I/O)	clk	0x00001020	0x0000102F	
☑	fpaa_active	PIO (Parallel I/O)	clk	0x00001030	0x0000103F	
☑	sdram	SDRAM Controller	clk			
	s1	Slave port		0x00800000	0x00FFFFFF	
☑	spi_master	SPI (3 Wire Serial)	clk			
	spi_control_port	Slave port		0x00001000	0x0000101F	0

図4　制御用FPGA（マイコン）の構成
SOPC Builderで設定した．CPUコア，SPIコア，パラレルI/O，EPCSコントローラ，SDRAMコントローラ，JTAG-UARTコアを使っている．回路規模は2,796LE（EP1C3の96％）になった．

▶ **図5**
FPAAの外部回路
マルチプレクス入力に与える分圧用の抵抗だけである．



▼ **図6　FPAAの内部回路**
マルチプレクス入力セルからの配線だけである．



Nios IIには，性能や回路規模の異なる3種類のコアが用意されています．今回は，もっとも回路規模が小さいNios II/eを用いました．

FPAAとのインターフェースのため，SPIコアは**図3**のように設定します．また，FPAAへのリセット出力と，コンフィグレーション完了を示す信号用のFPAAからの入力を1ビットずつ使用するためにI/Oを使います．このほか，EPCSコントローラ，SDRAMコントローラ，JTAG-UARTコアを使っています（**図4**）．回路規模は2,796ロジック・エレメント（EP1C3の96％）となりました．SDRAMのためにPLLも使っています．

● アナログ回路（FPAA）の設計

今回のテスト回路で使うのは，FPAA（AN220E04）の基本回路以外では，マルチプレクス入力に与える分圧用の抵抗だけです（**図5**）．

電源は＋5Vです．このため，I/Oが3.3VのFPGAとのインターフェースではレベル変換が必要です．

FPAA内部の回路設計は，開発ツールのAnadigmDesigner2で行います．今回はマルチプレクス入力セルのラインにBを選択するだけです（**図6**）．完成したら，まず，Anadigm Designer 2の「Dynamic Config」→「Algorithmic method...」で再構成の方式を設定します．次に，「Configure」→「Write configuration data to file...」を選択してC

リスト1　ヘッダ・ファイルの指定と変数の定義

```
#include <system.h>
#include <stdio.h>
#include “unistd.h”
                    //usleep
#include
“altera_avalon_pio_regs.h”
                    //IO
#include
        <altera_avalon_spi.h>
                    //spi
#include <ApiCode.h>
                    //FPAA
#include <CAMCode.h>
                    //FPAA

an_Byte *PrimaryConfigPtr;
an_Byte *ConfigPtr;
int ConfigSize;
int dcnt = 0;
```

リスト2　初回のコンフィグレーションのためのCソース・コード

```
// リセット
IOWR_ALTERA_AVALON_PIO_DATA( FPAA_RST_N_BASE, 0 );
usleep(10000);
IOWR_ALTERA_AVALON_PIO_DATA( FPAA_RST_N_BASE, 1 );

//回路データの取得
PrimaryConfigPtr = (an_Byte *)an_GetPrimaryConfigData(an_FPAA1, &ConfigSize);

//コンフィグレーション
alt_avalon_spi_command(SPI_MASTER_BSE,0,5,0,0,0,0);
alt_avalon_spi_command(SPI_MASTER_BASE,0,ConfigSize,PrimaryConfigPtr,0,0,0);
alt_avalon_spi_command(SPI_MASTER_BASE,0,1,0,0,0,0);
```

```
//書き換え用データの初期化
an_InitializeReconfigData(an_FPAA1);

//マルチプレクス入力セルの‘A’を選択
 an_setPadSelect(an_FPAA1_InputCell4, ‘A’);

//書き換えデータの取得
ConfigPtr = (an_Byte *)an_GetReconfigData(an_FPAA1, &ConfigSize);

//書き換え
alt_avalon_spi_command(SPI_MASTER_BASE,0,ConfigSize,ConfigPtr,0,0,0);
alt_avalon_spi_command(SPI_MASTER_BASE,0,1,0,0,0,0);

//書き換えデータをクリア
an_ClearReconfigData(an_FPAA1);
```

▶ リスト3　部分リコンフィグレーションのためのCソース・コード

ソース・コードを生成します．デフォルトでは，ApiCode.c，ApiCode.h，CAMCode.c，CAMCode.hの四つのファイルが生成されます．

部分リコンフィグレーションの制御方法

生成されたソース・コードをNios II IDEに追加し，コンパイルを行います．メイン・プログラムで必要なヘッダ・ファイルの指定と変数の定義を**リスト1**に示します．コンパイル時にエラーになったときには，これらの定義を忘れている可能性があります．

● 初回のコンフィグレーション

Nios IIによるFPAAコンフィグレーションは非常に簡単です．Cソース・コードを**リスト2**に示します

まず，FPAAをリセットします．リセットしなくても問題ないようですが，確実に動作させるためにはリセットしたほうがよいでしょう．

コンフィグレーションは，回路データを取得し，データを転送するだけです．ただし，転送前に40クロック，転送後に8クロックのダミー・クロックが必要です．

● 回路の部分リコンフィグレーション

それではいよいよ，マルチプレクス入力セルのパッドを切り替える“部分リコンフィグレーション”をテストします．初回のコンフィグレーションが正常に完了している状態で，**リスト3**を実行します．

まず，書き換え用のデータを初期化します．次に，マルチプレクス入力セルを指定します．ここではラインAを選択するものとします．そして，書き換え用の回路データを取得し，データを転送します．最初のコンフィグレーションのときに必要だった転送前の40クロックは書き換え時には必要ありません．転送後の8クロックのダミーは必要です．最後に，書き換えデータをクリアします．

ソフトウェアを実行することで，マルチプレクス入力セルが切り替わり，出力の変化が確認できます．

のむら・ひでまさ
(株)エイアンドティー

<筆者プロフィール>
野村英方．1998年にエイアンドティーに入社．以来，医療用分析装置のボード設計を中心とした設計業務に従事．

3